11. SQL, noSQL Injection

🛡️ SQL Injection & NoSQL Injection

1. **What is SQL Injection?**

- SQL Injection (SQLi) is a **code injection technique** where attackers insert malicious SQL statements into input fields, tricking the database into executing unintended commands.
- This allows attackers to **read, modify, or delete data**, bypass authentication, or execute administrative operations on the database.

2. **How Does NoSQL Injection Differ?**

- NoSQL Injection targets **NoSQL databases** (e.g., MongoDB, CouchDB) which use **JSON-like queries** instead of SQL syntax.
- Injection exploits occur by manipulating query objects or commands, often by injecting JSON or JavaScript snippets to alter query logic.
- Unlike SQLi which uses SQL commands, NoSQLi abuses **query operators and expressions** in NoSQL languages.

3. **What Are the Risks of SQL Injection?**

- Unauthorized access to sensitive data (e.g., user credentials, personal info).
- Data corruption or deletion (destructive queries).
- Bypassing authentication and authorization controls.
- Gaining shell access via database stored procedures.
- Lateral movement within networks if the database has elevated privileges.
- Complete system compromise in worst cases.

4. **Describe a UNION Attack**

- A **UNION SQL Injection attack** exploits the `UNION` SQL operator to **combine results from multiple SELECT statements**.
- An attacker appends a malicious query using `UNION` to retrieve data from different tables not normally accessible.
- This technique reveals data from other tables by merging it with the original query's result.

5. **Explain Blind SQL Injection**

- In **Blind SQL Injection**, the application does not directly display SQL errors or output, making it harder to exploit.

- Attackers infer information by sending payloads and analyzing application behavior (e.g., response time, content differences).
- Two types: **Boolean-based** (true/false responses) and **Time-based** (delays indicating query result).

---

6. How to Prevent SQL Injections?

- Use **Parameterized Queries** (Prepared Statements) to separate SQL code and data.
- Employ **Stored Procedures** that don't concatenate user input.
- Apply **input validation and sanitization** to ensure only expected data is processed.
- Implement least privilege for database accounts.
- Use Web Application Firewalls (WAFs) to detect and block injection attempts.
- Regularly patch and update database software.

---

7. What is a Parameterized Query?

- A **Parameterized Query** uses placeholders (e.g., `?` or named parameters) for user input, binding values separately from the SQL command.
- Prevents user input from being interpreted as executable code, mitigating injection risks.

---

8. What Are Stored Procedures in SQL?

- Stored Procedures are **precompiled SQL statements stored in the database**.
- They execute with predefined logic and accept parameters.
- When used properly, they can prevent direct concatenation of user inputs, reducing injection exposure.

---

9. Why is Input Validation Important?

- Validates user inputs against expected formats (e.g., length, type, allowed characters).
- Prevents malicious input from entering the system.
- Acts as the first defense layer before queries reach the database.
- Protects against injection and other attack vectors.

---

10. How Does NoSQL Injection Occur in MongoDB?

- Happens when user input is directly embedded in JSON queries without validation or sanitization.
- Example: injecting `{ "$gt": "" }` can modify query logic to bypass authentication or return unintended data.
- Can also exploit `$where` operator to execute arbitrary JavaScript code inside queries.

---

11. What is the Role of ORMs in Preventing Injections?

- ORMs (Object-Relational Mappers) abstract database queries through code.

- Good ORMs use parameterized queries internally, reducing injection risks.
- However, improper use (e.g., raw queries, string concatenation) still exposes risks.

---

12. **Can NoSQL Databases like MongoDB Be Injected?**

- Yes, NoSQL databases are **not immune**. Injection is possible via unsanitized inputs altering query structure or injecting JavaScript code.
- Prevention involves strict input validation, avoiding direct concatenation, and disabling unsafe query operators.

---

13. **What is Escaping User Input in SQL Queries?**

- Escaping means **prefixing special characters** in input (like quotes) with escape characters to neutralize their meaning.
- Helps prevent injection but is less reliable than parameterized queries, as escaping can be bypassed if incomplete.

---

14. **Explain the Use of LIMIT in SQL Injection Attacks**

- Attackers use `LIMIT` clause in injection payloads to **control the number of returned rows**.
- Useful to extract data row-by-row during blind or error-based attacks to avoid overwhelming output.

---

15. **How to Use Regular Expressions for Input Validation?**

- Regex can enforce **strict patterns** (e.g., only alphanumeric, specific length, no special chars).
- Essential to filter and sanitize inputs before they reach the database query.
- Helps block malicious payloads but should be combined with parameterized queries for defense in depth.

---

16. **What is a NoSQL Injection Attack Vector?**

- Injection through manipulation of query operators like `$gt`, `$ne`, `$where`.
- Examples include injecting code into JSON objects, exploiting unsafe parsing, or abusing JavaScript execution features in NoSQL databases.
